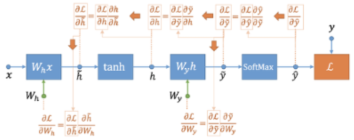


Figure 2: Schematic view of the *backpropagation* on a neural network. We start by calculating the derivative of the cost $\mathcal{L}$ compared to the output ($\hat{y}$ on the diagram) then we go back in the network by reusing the derivatives calculated previously during the calculation of the new derivatives.
Note: in the vector case, it is necessary to add sums as indicated in the equation (2).

layer can be reused to calculate the gradients with respect to the input and the parameters of this same layer: it suffices to multiply it by a simple local gradient (from the output relative to input or parameters).

The successive calculation of the gradients of the different layers is called the *backward* pass.

Questions

10. What seem to be the advantages and disadvantages of the various variants of gradient descent between the classic, mini-batch stochastic and online stochastic versions? Which one seems the most reasonable to use in the general case?
11. ★ What is the influence of the *learning rate* η on learning?
12. ★ Compare the complexity (depending on the number of layers in the network) of calculating the gradients of the *loss* with respect to the parameters, using the naive approach and the *backprop* algorithm.
13. What criteria must the network architecture meet to allow such an optimization procedure ?
14. The function SoftMax and the *loss* of *cross-entropy* are often used together and their gradient is very simple. Show that the *loss* can be simplified by:
$$\ell = -\sum_i y_i \tilde{y}_i + \log\left(\sum_i e^{\tilde{y}_i}\right).$$
15. Write the gradient of the *loss* (*cross-entropy*) relative to the intermediate output $\tilde{y}$
$$\frac{\partial \ell}{\partial \tilde{y}_i} = \dots \quad \Rightarrow \quad \nabla_{\tilde{y}}\ell = \begin{bmatrix} \frac{\partial \ell}{\partial \tilde{y}_1} \\ \vdots \\ \frac{\partial \ell}{\partial \tilde{y}_{n_y}} \end{bmatrix} = \dots$$
16. Using the *backpropagation*, write the gradient of the *loss* with respect to the weights of the output layer $\nabla_{W_y}\ell$. Note that writing this gradient uses $\nabla_{\tilde{y}}\ell$. Do the same for $\nabla_{b_y}\ell$.
$$\frac{\partial \ell}{\partial W_{y,ij}} = \sum_k \frac{\partial \ell}{\partial \tilde{y}_k}\frac{\partial \tilde{y}_k}{\partial W_{y,ij}} = \dots \quad \Rightarrow \quad \nabla_{W_y}\ell = \begin{bmatrix} \frac{\partial \ell}{\partial W_{y,11}} & \cdots & \frac{\partial \ell}{\partial W_{y,1n_h}} \\ \vdots & \ddots & \vdots \\ \frac{\partial \ell}{\partial W_{y,n_y1}} & \cdots & \frac{\partial \ell}{\partial W_{y,n_yn_h}} \end{bmatrix} = \dots$$
17. ★ Compute other gradients : $\nabla_{\tilde{h}}\ell$, $\nabla_{W_h}\ell$, $\nabla_{b_h}\ell$.

14) $\ell(y,\hat{y}) = -\sum_{i=1}^{n_y} y_i \cdot \log(\hat{y}_i)$ $\hookrightarrow$ $\hat{y} = \text{softmax}(\tilde{y}) =$

$$= -\sum_{i=1}^{n_y} y_i \log\left(\frac{e^{\tilde{y}_i}}{\sum_{j=1}^{n_y} e^{\tilde{y}_j}}\right) =$$
$$= -\sum_{i=1}^{n_y} y_i \left[\log(e^{\tilde{y}_i}) - \log\left(\sum_{j=1}^{n_y} e^{\tilde{y}_j}\right)\right]$$
$$= -\sum_{i=1}^{n_y} y_i \tilde{y}_i + \left(\sum_{i=1}^{n_y} y_i\right) \cdot \log\left(\sum_{j=1}^{n_y} e^{\tilde{y}_j}\right)$$
$$= -\sum_{i=1}^{n_y} y_i \tilde{y}_i + \log\left(\sum_{j=1}^{n_y} e^{\tilde{y}_j}\right)$$

$\hookrightarrow$ This is the loss !!

15)
$$\ell = \quad \nabla \tilde{y}_1 = -y_1 + \frac{e^{\tilde{y}_i}}{\sum_{k=1}^{n_y} e^{\tilde{y}_k}}$$

Softmax !

$$= -y_i + \hat{y}_i = \hat{y}_i - y_i$$

16)
$$\frac{\partial \ell}{\partial W^y_{i,j}} = \sum_{k=1}^{n_y} \underbrace{\frac{\partial \ell}{\partial \tilde{y}_k}}_{\text{we know !! Eq (1) !!}} \cdot \frac{\partial \tilde{y}_k}{\partial W^y_{ij}}$$

$$\frac{\partial \tilde{y}_k}{\partial W^y_{ij}} = \left([W^y h]_k\right) = \frac{\partial \tilde{y}_k}{\partial W^y_{ij}}\left(\sum_{p=1}^{n_h} W_{kp} h_p\right) =$$
$$= \begin{cases} h_j & \text{iff } i = k \\ 0 & \text{otherwise} \end{cases}$$

Real-life !!

$\tilde{y} = W^y h + b^y$

$h (W^y)^T$

Real life !! feature

Sample $\boxed{h}$

Combined !

$\hookrightarrow$
$$\frac{\partial \ell}{\partial W^y_{i,j}} = \sum_{k=1}^{n_y} \frac{\partial \ell}{\partial \tilde{y}_k} \cdot \frac{\partial \tilde{y}_k}{\partial W^y_{ij}}$$
$$\hookrightarrow \quad = (\hat{y}_i - y_i) \cdot h_j$$

$$\nabla_{W^y} \ell = \begin{bmatrix} (\hat{y}_1 - y_1) \cdot h_1 & (\hat{y} - y_1) \\ (\hat{y}_{n_y} - y_{n_y}) h_y & (\hat{y}_{n_y} - y_{n_y}) \end{bmatrix}$$

$$\frac{\partial \ell}{\partial b_i^y} = \sum_{k=1}^{n_y} \underbrace{\frac{\partial \ell}{\partial \tilde{y}_k}}_{\text{we know !!}} \cdot \frac{\partial \tilde{y}_k}{\partial b_i^k} \ ?$$

$$\frac{\partial \tilde{y}_k}{\partial b_i^y} = \frac{\partial [W^y h + b^y]_k}{\partial b_i^y}$$
$$= \frac{\partial (b^y_k)}{\partial b_i^y} = \begin{cases} 1 & \text{if } i = k \\ 0 & \text{otherwise} \end{cases}$$

So :
$$\frac{\partial \ell}{\partial b_i^y} = \frac{\partial \ell}{\partial \tilde{y}_i} = \hat{y}_i - y_i$$

$$\frac{\partial \ell}{\partial h_i} = \sum_{k=1}^{n_y} \frac{\partial \ell}{\partial \tilde{y}_k} \frac{\partial \tilde{y}_k}{\partial h_i}$$

$$\frac{\partial \tilde{y}_k}{\partial h_i} = \frac{\partial([W^y h + b^y]_k)}{\partial h_i} = \frac{\partial [W^y h]_k}{\partial h_i} =$$
$$= \frac{\partial\left(\sum_{p=1}^{n_h} W^y_{k,p} h_p\right)}{\partial h_i} = W^y_{ki}$$

$$\frac{\partial \ell}{\partial h_i} = \sum_{k=1}^{n_y} \frac{\partial \ell}{\partial \tilde{y}_k} \cdot \frac{\partial \tilde{y}_k}{\partial h_i}$$
$$\frac{\partial \ell}{\partial h_i} = \sum_{k=1}^{n_y} (\tilde{y}_k - y_k) \cdot W^y_{k,i}$$

Section 2 – Implementation

We now have all the equations allowing us to make predictions (*forward*), to evaluate (*loss*) and to learn (*backward* and gradient descent) our model.

We are now going to implement this network with PyTorch. The objective of this part is to gradually familiarize yourself with this framework.

This part is inspired by the PyTorch getting started tutorial available at `http://pytorch.org/tutorials/beginner/pytorch_with_examples.html`. Do not hesitate to watch it in parallel with this lab to help you write the requested functions.

2.1 Forward and backward manuals

We will start by coding the neural network by simply using basic mathematical operations and therefore directly transcribing our mathematical equations. We will use the functions found in the `torch` : `https://pytorch.org/docs/1.2.0/index.html` package.

In torch, data is stored in objects of type `torch.Tensor`, equivalent to `numpy.array`. The basic PyTorch functions return objects of this type.

1. **★ ★ Discuss and analyze your experiments following the implementation. Provide pertinent figures showing the evolution of the loss; effects of different batch size / learning rate, etc.**
2. Write the function `init_params(nx, nh, ny)` which initializes the weights of a network from the sizes n_x, n_h and n_y and stores them in a dictionary. All weights will be initialized according to a normal distribution of mean 0 and standard deviation 0.3.
   Hint: use the `torch.randn` and `torch.zeros` functions.
3. Write the function `forward (params, X)` which calculates the intermediate steps and the output of the network from an input batch `X` of size $n_{batch} \times n_x$ and weights stored in `params` and store them in a dictionary. We return the dictionary of intermediate steps and the output $\hat{Y}$ of the network.
   Hint: we will use `torch.mm` for matrix multiplication, and `torch.tanh`, `torch.exp`, `torch.sum`
4. Write the function `loss_accuracy (Yhat, Y)` which computes the cost function and the precision (rate of good predictions) from an output matrix $\hat{Y}$ (output of `forward`) with respect to a ground truth matrix Y of the same size, and returns the *loss* `L` and the precision `acc`.
   Note: We will use the `_, indsY = torch.max (Y, 1)` function which returns the index of the predicted class (or to be predicted) for each example.
   Hints: `torch.mean`, `torch.max`, `torch.log`, `torch.sum`
5. Write the function `backward (params, outputs, Y)` which calculates the gradients of the *loss* with respect to the parameters and stores them in a dictionary.
6. Write the function `sgd (params, grads, eta)` which applies a stochastic gradient descent by mini-batch and updates the network parameters from their gradients and the learning step .
7. Write the global learning algorithm using these functions.

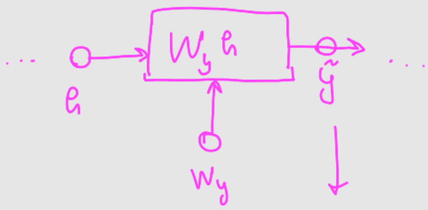
16)

$$\boxed{\nabla e_{W_y}} = \frac{\partial e}{\partial W_{y,ij}} = \frac{\partial e}{\partial \tilde{y}} \cdot \frac{\partial \tilde{y}}{\partial W_y} =$$

VECTOR ~~CASE~~

$$= \sum_{k=1}^{n_y} \frac{\partial e}{\partial \tilde{y}_k} \, \frac{\partial \tilde{y}_k}{\partial W_{y,ij}} =$$

we know, see answer 15



$\tilde{y} = W_y h + b_y$

$$\frac{\partial \tilde{y}_k}{\partial w_{y,ij}} = \frac{\partial([W^y_k h + b^y])_k}{\partial W_{y,ij}} = \frac{\partial \tilde{y}_k}{\partial w_{y,ij}} \left(\sum_{e=1}^{n_h} W_{ke} h_e \right) =$$

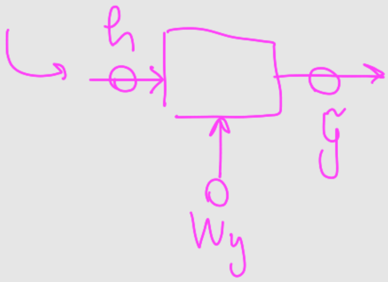
$$= \begin{cases} h_j & \text{if } i=k \\ 0 & \text{otherwise} \end{cases}$$

$\Downarrow$

$(\hat{y}_i - y_i)$ $\quad h_j$

$$\frac{\partial e}{\partial W_{y,ij}} = \sum_{k=1}^{n_y} \frac{\partial e}{\partial \tilde{y}_k} \cdot \frac{\partial \tilde{y}_k}{\partial W_{y,ij}} = (\hat{y}_i - y_i) \cdot h_j$$

$$\boxed{\nabla_{b_h} e} = \frac{\partial e}{\partial b_i^y} = \sum_{k=1}^{n_y} \frac{\partial e}{\partial \hat{y}_k} \cdot \frac{\partial \tilde{y}_k}{\partial b_i^k}$$



$$\frac{\partial \tilde{y}_k}{\partial b_i^y} = \frac{\partial \left[W^y \cdot h + b^y\right]_k}{\partial b_i^y} = \frac{\partial (b_k^y)}{\partial b_i^y} =$$

$$= \begin{cases} 1 & \text{if } i=k \\ 0 & \text{otherwise} \end{cases}$$

$$\Downarrow$$

$$\frac{\partial e}{\partial b_i^y} = \frac{\partial e}{\partial \tilde{y}_i} = \hat{y}_i - y_i$$

17)

$$\boxed{\nabla_{\tilde{h}} e} = \frac{\partial e}{\partial h_i} = \sum_{k=1}^{n_y} \frac{\partial e}{\partial \tilde{y}_k} \cdot \frac{\partial \tilde{y}_k}{\partial h_i} \; *$$

$$* \quad \frac{\partial \tilde{y}_k}{\partial h_i} = \frac{\partial([W^y h + b^y]_k)}{\partial h_i} = \frac{\partial [W^y \cdot h]_k}{\partial h_i} =$$

$$= \frac{\partial \left(\sum_{p=1}^{n_h} W^y_{k,p} \cdot h_p \right)}{\partial h_i} = W^y_{k,i}$$

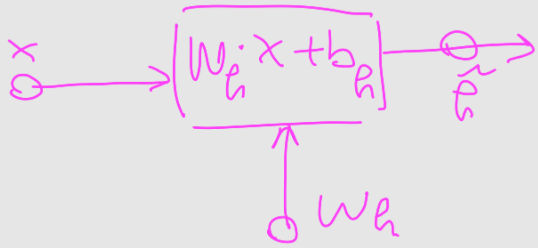
$$\Downarrow$$

$$\frac{\partial e}{\partial h_i} = \sum_{k=1}^{n_y} \frac{\partial e}{\partial \tilde{y}_k} \cdot \frac{\partial \hat{y}_k}{\partial h_i} =$$

$$= \sum_{k=1}^{n_y} (\hat{y}_k - y_k) \cdot W^y_{k,i}$$

$$\boxed{\nabla_{W_h} e} = \frac{\partial e}{\partial W_h} = \frac{\partial e}{\partial \tilde{h}} \cdot \frac{\partial \tilde{h}}{\partial W_h} =$$

$$= \sum_{k=1}^{n_h} \frac{\partial e}{\partial \tilde{h}_k} \cdot \frac{\partial \tilde{h}_k}{\partial W_h} \; {}^{*}$$



LOOK PREVIOUS ANSWER

$$* \quad \frac{\partial \tilde{h}_k}{\partial W_h} = \frac{\partial \left([W^h x + b^h]_k \right)}{\partial W_h} = \begin{cases} x_j \; ? & \text{if } i=k \; ? \\ 0 & \text{otherwise} \end{cases}$$

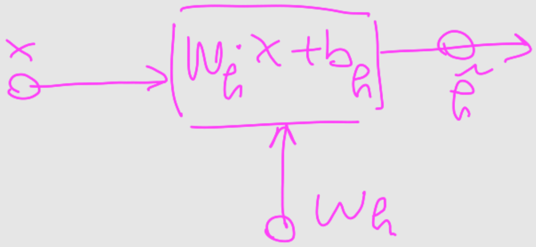
$$\Downarrow$$

$$\frac{\partial e}{\partial W_h} = \sum_{k=1}^{n_y} (\hat{y}_k - y_k) \cdot W^y_{k,i} \cdot x_i$$

$$\boxed{\nabla e_{b_h}} = \frac{\partial e}{\partial b_h} = \frac{\partial e}{\partial \tilde{h}} \circ \frac{\partial \tilde{h}}{\partial b_h} =$$

$$= \sum_{k=1}^{n_h} \frac{\partial e}{\partial \tilde{h}_k} \cdot \frac{\partial \tilde{h}_k}{\partial b_h} \; *$$

LOOK PREVIOUS ANSWER



$$* \quad \frac{\partial \tilde{h}_k}{\partial b_h} = \frac{\partial\left([W^h x + b^h]_k\right)}{\partial b_h} = \begin{cases} b_i & \text{if } i = k \\ 0 & \text{otherwise} \end{cases}$$

$$\Downarrow$$

$$\frac{\partial e}{\partial W_h} = \sum_{k=1}^{n_y} (\hat{y}_k - y_k) \cdot W^y_{k,i} \cdot b_i$$